Interleaving Granularity on High Bandwidth Memory Architecture for CMPs

Felipe Cabarcas*‡, Alejandro Rico*, Yoav Etsion* and Alex Ramirez*†
*Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politecnica Catalunya, Barcelona, Spain
‡Universidad de Antioquia, Medellin, Colombia
Email: {felipe.cabarcas, alejandro.rico, yoav.etsion, alex.ramirez}@bsc.es

***Abstract*—Memory bandwidth has always been a critical factor for the performance of many data intensive applications. The increasing processor performance, and the advert of single chip multiprocessors have increased the memory bandwidth demands beyond what a single commodity memory device can provide. The immediate solution is to use more than one memory device, and interleave data across them so they can be used in parallel as if they were a single device of higher bandwidth.**

In this paper we showed that fine-grain memory interleaving on the evaluated many-core architectures with many DRAM channels was critical to achieve high memory bandwidth efficiency. Our results showed that performance can degrade up to 50% due to achievable bandwidths being far from the maximum installed.

I. Introduction

Power consumption and design complexity have led the computer architecture community to design chip multiprocessors (CMP). Current commercial CMPs integrate between 4 and 8 processors in one chip, but the current interpretation of Moore's Law says that the number of cores will double every 18 months, leading to chips with 16 to 32 cores in the next generations.

Memory bandwidth has always been a critical performance aspect, even for single processor architectures. The increased compute performance provided by smarter architectures, deeper pipelines, and higher clock frequencies has reduced computation time to the point where more and more applications have turned from compute-bound to memory-bound. The problem is that computing has become so fast, that data can not be read from memory fast enough to feed the processor's functional units.

Most computer systems today are built from commodity components, and that includes high-ranging supercomputers in the Top500 list like the Roadrunner at Los Alamos National Laboratory. This implies that whatever memory system is designed, it has to work with standard off-the-shelf memory devices, like current DDR2 and DDR3 DIMMs. The memory bandwidth provided by one DIMM is actually fixed by technology and JEDEC standards. Top specifications for DDR2 memory offer 16 bytes / cycle at 533 MHz for a total of 8.533 GB/s. Top specifications for DDR3 offer 16 bytes / cycle at 800 MHz, a total of 12.8 GB/s.

However, such maximum bandwidth can only be achieved under ideal conditions like a sequential access pattern (stride 1 accesses), or an access pattern that can be optimally distributed across the available banks. Optimizing data stream distribution across memory banks has been a great concern from the early days of supercomputing, when the memory system was the most expensive part of the dominant vector computers like the Cray-1 or the Cray-YMP.

The use of chip multiprocessors only increases the pressure on the memory bandwidth, since there are more processors reading from a shared memory system. There have been reports by Sandia National Labs of CMP systems becoming slower instead of faster after a number of cores have been integrated due to lack of memory bandwidth [1]. The Sandia press release reports insignificant performance gains when going from 4 to 8 cores, and performance slowdowns when going to 16 cores. They explicitly identify the need to design memory systems that "provide a dramatic improvement over what was available 12 months ago".

As the number of processors increases, the bandwidth offered by a top DDR3 DIMM is not enough. It becomes necessary to use more than one DIMM, and organize them so they offer the same performance of an ideal faster device.

In this paper, we evaluate multiple memory system organizations targeting high performance computing applications running on a shared memory address space architecture with 8 and 32 on-chip processors each with its private scratchpad memory, as a scaled-up Cell/B.E. [2]. The applications have been written in StarSs [3], a task-level data-flow programming model similar to Cilk [4], RapidMind [5], Sequoia [6], and Tflux-DDM [7]. These programming models let the programmer write a seemingly sequential program, and annotate the input and output parameters of functions that can potentially execute as parallel tasks. While the program executes on a master processor, the runtime system offloads task computation to the worker processor and their data is transferred to the scratchpad memories ahead of time, overlapping the data transfer with the computation of the previous task.

This research is supported by the Consolider contract number TIN2007-60625 from the Ministry of Science and Innovation of Spain, the European Network of Excellence HIPEAC-2 (ICT-FP7-217068), the ENCORE project (ICT-FP7-248647) and the IBM-BSC MareIncognito project. Cabarcas was also supported in part by the Program AlBan, the European Union Program of High Level Scholarships for Latin America (scholarship No. E05D058240CO).

978-1-4244-7938-2/10/$26.00 ©2010 IEEE

The objective of the paper is to show the importance of the interleaving of the storage scheme to extract close to peak bandwidth of the installed memory modules.

The rest of this paper is organized as follows. First, we discuss related work in the area in Section II. In Section III we present the evaluated memory system architectures. Then, in Section IV we describe the applications under study, and characterize the memory traffic that is generated. We discuss the performance evaluation results in Section V. Finally, in Section VI we present our conclusions regarding the design of a next-generation CMP memory system.

II. Related Work

The CMP architecture considered in this work is similar in concept to a vector multi-processor. Processors in the CMP access their local storage for processing (vector registers) and data is loaded in such local memory through DMA GET/PUT operations (like vector LOAD/STORE with stride 1). This means that there are some similarities in the access pattern and memory usage of these architectures. In order to design the memory architecture of such a CMP, we must first review the memory system design of vector processors.

The mechanism that distributes accesses across the installed memory modules is called storage scheme. The literature on vector processors describes three main storage schemes: interleaving, skewing [8], and linear transformations [9]. These basic schemes can provide conflict-free access for simple vector stride access patterns, like the stride 1 access of DMA operations.

More complex stride access patterns can be optimized with variations of the interleaving scheme, as proposed by Harper III et al. [10]. Other strategies are required for conflict free accesses using general strides, like XOR strategies in [11], later extended in [12] to cope with more strides and unmatched memories. However all these papers consider only a single vector processor.

A block interleaving method for conflict-free access on vector multiprocessors is proposed in [13]. However, this scheme assumes a matched system where the number of memory sections (memory channels) is the same as the number of processors, and the total number of memory modules (banks) is equal or greater than the number of processor multiplied by the memory latency. In our current environment, where we consider 32 on-chip processors, and hundreds of cycles of memory latency, this proposal is simply not practical.

Some of these ideas have been used in the general purpose environment to improve the memory efficiency in systems with caches. In [14], [15] it is shown that XOR address mapping can be used to randomize bank ordering access in cache write backs, maintaining the locality properties of the applications to access the row buffers on the DRAMs (contiguous row ordering). This allows the memory scheduler to use the banks more efficiently and therefore improve bandwidth. The CMP architecture considered in this paper does not consider a cache between the processors and the memory, therefore this strategy is not applicable.

XOR strategies are used to reduce conflicts on the installed devices and increase utilization, however, they can not change the granularity of the access pattern. Therefore, they can be used in addition to the interleaving granularity used in the system. In this paper, we do not consider any XOR strategy since they would increase the complexity of the system and blur the interleaving granularity effects.

The interleaving strategy followed in [16] also seeks to improve the performance of the memory systems with caches. It uses the cache index bits to select a DRAM page and bank, so that in cache write backs the addresses will fall in the same bank and page. The problem with this strategy is that it will not benefit from the spatial locality opportunities for the open row policy of the DRAM. However, its simplicity, and the fact that the architecture considered in this paper does not contain caches, makes it a candidate to be tested.

Very aggressive channel and bank address mappings are proposed and used in [17], [18], [19]. The idea of these mappings is to distribute contiguous addresses on different channels in order to access them in parallel when consecutive addresses arrive to the memory controller, even if some opportunities of row locality on the DRAMs are lost, these are designed for maximum bandwidth efficiency use.

Given the simplicity of interleaving approaches, they are used as the starting point of our work. We extend previous work by examining the policies in the context of large scale CMP architectures where multiple address streams must be interleaved for fairness, and the pressure on the memory bandwidth is higher.

The aggressiveness of address mappings in this paper is measured in terms of the placement of consecutive address words (128B) on the installed channels and banks. Considering that each DRAM page is 1KB (8 x 128 byte words), the less aggressive address mappings will assign consecutive words to the same channel and bank, while the more aggressive ones would spread consecutive words across different channels and banks.

The second important aspect of the memory controller design is the memory scheduler. It was shown in [12], [20] for vector processors and in [21], [22] for streaming applications, that out-of-order servicing of accesses is a critical factor. Later, [23] shows the importance of memory reordering in web applications. In particular, they show that having a queue per bank and aggressive command reordering obtains the best bandwidth efficiency.

Ipec et al. [24] show that a CMP memory controller, on a 4-core (8-thread) processor, should adapt dynamically to the characteristics of the applications. For such CMPs, they show that average data bus utilization for the best static controller only reaches 46 percent efficiency, and even an optimistic memory controller only achieves 80 percent.

Other works like [25] propose a memory controller for vector processors that uses vector commands directly, instead of individual loads and stores. This allows the controller to reorder individual DRAM commands to efficiently exploit their locality properties.

Similarly, Mckee et al. [26], [27] show that close to 100 percent efficiency can be obtained if the access pattern is known, or if the memory requests are organized such that the memory scheduler can choose from them properly.

In this paper we consider a memory scheduler that chooses the first ready request from a separate queue per channel, similar to the one described in [22], [23]: "To maximize request concurrency, the lowest-order bits after the DRAM page offset choose the DRAM channel, the next bits choose the bank, and the highest-order bits choose the row. Address bits are assigned so that the most significant bits identify the smallest-scale component in the system, and the least significant bits, which should change most often from request to request, identify the largest-scale component in the system." [28].

III. Memory System Architecture

Given the extreme computation power available on current CMPs, and the low bandwidth of current commodity DRAM DIMM modules, it has been necessary to include more independent ports to memory. That is, it has been necessary to add multiple DIMM channels to the on-chip memory controller. However, in order to allow parallel applications to benefit from the extra bandwidth offered by the extra channels, it is necessary to use an adequate memory interleaving policy so channels are actually accessed in parallel.

In this section we describe the memory system architecture employed in our evaluations, and the different memory interleaving policies evaluated.

A. Memory architecture

Figure 1 shows the memory system architecture evaluated in this paper. The figure shows 32 processors, distributed in 4 clusters, and 2 memory interface controllers (MICs) connected to a hierarchical Network-on-Chip (NoC). Each one of the MICs manages up to 4 off-chip DDR2 or DDR3 memory devices (DIMM). Each one of the links in the figure is capable of transmitting 8 bytes per cycle at 3.2 GHz. That is, all links provide 25.6 GB/s bandwidth. Dual-MIC configurations using 8 DRAM channels can be found in some existing processors, like the IBM POWER6 [29]. The processor architecture considered, which is basically a scaled-up Cell/B.E., does not include a cache hierarchy. Therefore, all processor requests go to memory and there is no filtering effect from caches, which facilitate the measurement of the achievable bandwidth in main memory.

The NoC allows two simultaneous communications, as long as they do not traverse the same links. For example, the figure shows a processor from the first cluster sending data to the first MIC, and a processor from the third cluster sending data to the second MIC in parallel. Accounting for two parallel data streams, the maximum bandwidth available is 51.2 GB/s. In order to achieve such 51.2 GB/s bandwidth, we use 8 DDR2-800 DIMMs of 6.4 GB/s each (4 of them connected to each MIC).

We consider two architectures configurations: 8 and 32 processors with 25.6GB/s and 51.2 GB/s peak memory bandwidth, respectively. We could evaluate memory systems with more MICs and DRAM channels but they may not be feasible implementations, as each DRAM channel requires over 100 off-chip pins, and that such pin count does not scale with technology like transistor density.

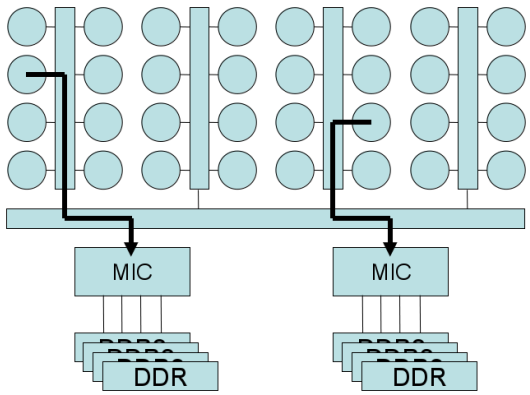


Fig. 1. 32 processors architecture with 2 on-chip memory controllers managing 4 DRAM devices each.

B. Storage schemes

Based on previous works as explained in section II, we use address-based interleaving policies as our memory storage scheme. The physical location of a datum is determined based on selected bits from its physical address. Those bits select the memory controller (M), the device channel (H), the bank within the device (B), the row within the bank (R), and the column within the row (C). Not all the bits used for any of the parts need to be consecutive in the datum's address.

DRAM memory cells are formed by capacitors and therefore, on read or write operations, a page (or line) is temporary stored on a buffer that it is used to restore (precharge) the values to the capacitors once the operation has finished. Most DRAMs have one of such buffers for each bank and it is usually of 1KB or 2KB (and the DIMM pages are 8KB or 16KB). There are two types of row operations supported by DRAM: autoprecharge (AP) and not-autoprecharge (NAP), also called *close* and *open* page operations respectively. In NAP, the accessed page is stored on a buffer so that future operations on the same page do not have to re-open the page before reading or writing. This strategy is preferred for high locality access patterns, but it would suffer when pages have to be closed constantly. In these cases, AP row operation is preferred.

Table I shows the interleaving policies (BHM-X) considered in this paper. The address mapping shows how the bits are used to select the location of each address on the different memory structures (MIC, channel, bank, row and column). The number of bits in x determines the interleaving granularity, i.e. the number of consecutive bytes mapped to a DRAM channel. For example, on a system with 2 MICs and 4 channels per MIC and a BHM-4096 interleaving strategy, given a DMA transaction of 16KB, it would access just 4 DRAM channels (assuming the initial address is aligned to 4KB) as the mapping changes the channel every 4KB.

Family	Address Mapping	Description
BHM-X	R:C:B:H:M:x	Every X= 2^x bytes the MIC and channel are changed

TABLE I
PHYSICAL ADDRESS BITS USED TO DETERMINE THE CORRESPONDING STRUCTURE: MIC (M), DRAM ROW (R), BANK (B), CHANNEL (H), AND COLUMN (C,x).

IV. APPLICATIONS

The application traces used in this paper were generated on a Cell/B.E. running at 3.2GHz. The traces consist of all the workers' DMA transfers and their timing information, as well as that of the CPU bursts (runtime, arrival time, etc.). The CPU burst times obtained on the Cell/B.E. are used as the baseline timings for both masters and workers. To minimize OS noise, each benchmark was executed multiple times and the fastest execution was selected as the representative trace.

To guarantee that changing the number of simulated processors does not break application semantics, traces also include inter-task dependency information. This information allows the simulator to group all tasks dispatched by the master in a single task list, and dynamically schedule them to simulated worker processors. The dependency information is required to verify the scheduling correctness, so that no task is scheduled before all its predecessors have finished — even though the scheduling order may differ from that of the original trace because of the increased number of worker processors.

For the experiments in this paper, we have selected six high-performance parallel applications in order to test the memory bandwidth offered by our architecture. All applications have been optimized using double buffering (overlap DMA traffic with task execution) in order to maximize the pressure on both the processors and the memory system. The six applications are:

1) **FFT3D**: Fast Fourier Transform of a three dimensional cube. The kernel transforms a 256x256x256 cube of complex numbers (2 floats), and performs (1) a FFT on each row (FFT1D), (2) a rotation of the cube, (3) a second FFT, (4) a second rotation, and (4) a third FFT.
2) **MatMul**: Blocked matrix multiplication of 4096x4096 float matrices.
3) **Cholesky**: Blocked Cholesky factorization of a 4096x4096 float matrix. The matrix is traversed by columns to perform the factorization.
4) **Knn**: k-Nearest Neighbors algorithm. A distance based object classification algorithm, featuring lazy learning. The problem size consists of 100000 samples, 16384 points to label, 48 dimensions, with 30 neighbors, and 20 classes.
5) **Kmean**: k-Means algorithm. A distance based data clustering algorithm that performs iterative refinements. The problem size is 256K points, 64 dimensions, 64 centers, with the threshold set to 0.01.
6) **CheckLU**: Blocked Sparse LU decomposition, followed by a matrix multiplication verifying that $A = L \times U$.

Table II shows a summary of the main characteristics of each application: number of tasks, average task runtime, memory footprint, and estimated bandwidth required per task. The bandwidth estimate was obtained from the average task data size and runtime. Given that not all tasks request the same amount of data, and have different duration, the average bandwidth was measured dividing total application bytes transferred and total task execution time.

The estimated bandwidth required per task clearly demonstrates how a limited memory bandwidth impedes the scalability of parallel architectures. For brevity, we remind the reader that the fastest DDR3 DRAM single channel configuration peaks at 12.8GB/s, and that 16 parallel matrix multiply tasks would already require 22.7GB/s.

Figure 2 shows the projection of the estimated bandwidth (from Table II) for different number of processors. The following section evaluates the performance of the configuration with 8 processors and a maximum bandwidth of 25.6GB/s and, the one with 32 processors and 51.2GB/s. It can be seen that, in the first case, only FFT3D requires over the maximum 25.6GB/s while the rest of applications require half or less than that. In the second case, the 51.2GB/s maximum bandwidth is less than half of the required by FFT3D and barely satisfies the bandwidth of Cholesky, Kmean and MatMul.

In the first scenario, achieving 50% of maximum bandwidth is enough for most applications. However, in the second scenario, any bandwidth efficiency loss will degrade the performance of most applications. Only Knn has a large margin between the required and the maximum bandwidths.

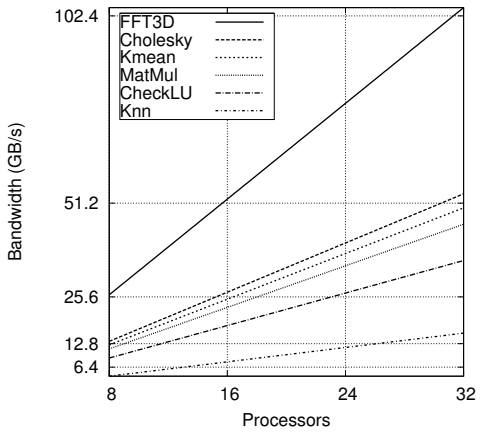


Fig. 2. Bandwidth requirement projection for different number of on-chip processors.

V. PERFORMANCE EVALUATION

For the performance evaluation of the different memory interleaving granularities for the presented applications we use a trace-driven cycle-accurate simulator TaskSim [30] of the architecture described in section III-A, using the parameters in table III.

TaskSim targets the simulation of parallel applications coded in a master-worker task offload computational model.

Kernel	Est. BW Per task	No. of tasks	Avg. Task runtime(μs)	Problem size(MB)	Task Block Size
FFT3D	3.27GB/s	32768	13.9	128	64x64 subMat. Trans., 256 line fft
MatMul	1.42GB/s	262144	25.8	192	64x64 subMat.
Cholesky	1.68GB/s	357760	28.0	512	64x64 subMat.
Kmean	1.56GB/s	335872	30.7	195	8192 vector
Knn	0.49GB/s	800768	7.9	36	8192 vector
CheckLU	1.11GB/s	54814	45.7	256	64x64 subMat.

TABLE II
APPLICATION MAIN CHARACTERISTICS.

Parameter	Value	Parameter	Value
Clock	3.2GHz	DRAMs	DDR3-800
NoC Ports	25.6GB/s	DRAM BW	6.4GB/s
DIMMs	4 & 8	Page policy	closed
MICs	1 & 2	t_{CL}	12.5ns
MIC Queue	512	t_{RAS}	37.5ns
MIC sched.	in order	t_{RC}	50ns
processors	8 & 32	t_{RCD}	12.5ns
NoC Latency	1cy	t_{RP}	21.5ns
NoC BW	25.6GB/s	t_{WR}	15ns
NoC Rings	crossbar	t_{WTR}	10ns
t_{Burst}	20ns	t_{DQSS}	5ns

TABLE III
BASELINE ARCHITECTURE PARAMETERS.

The traces contain timing information about the computation phases of the processors, such as task generation for the master processor or task execution for worker processors, DMA transfers, and the inter-task dependencies.

The computational CPU phases (bursts), such as task execution, are not simulated in detail. The burst duration is obtained from the trace file, and is simulated as a single instruction with the same runtime as the whole burst. Contrarily, the trace time for phases involving access to shared resources in the architecture, such as waiting for DMA transfers, are discarded, and their timing is simulated in a cycle-accurate way by means of detailed simulation of DMA controllers, caches, interconnection, memory controllers, and DRAM DIMMs.

Table IV shows the FFT3D timing break down for BHM-128 and BHM-4096, in an 8-processor, 1-MIC and 4-DRAM channel system. The application behaves similarly for both interleavings, except for the second transposition that takes 2.8 times longer with BHM-4096 than with BHM-128.

Figure 3 shows the number of requests per channel over time for the two interleaving strategies. The traffic that reaches the MIC on the BHM-128 is nicely distributed among the 4 channels; while in the BHM-4096 case, the requests use only one channel at a time. The consequence is that, in BHM-4096, the maximum effective bandwidth is close to that of a single DRAM channel, leading to the poor performance shown for the second transposition in Table IV.

These results show a case where the interleaving strategy has a major impact on performance for a particular access pattern due to its influence on the achievable bandwidth (bandwidth efficiency). The following figures show the impact on bandwidth efficiency and performance of the different interleaving granularities on the evaluated applications for 2 architectures: 8 processors with 25.6GB/s (Figure 4) and 32 processors with 51.2GB/s (Figure 5).

Figure 4(a) shows the speedup results for all applications against BHM-4096 (BHM-4K). The performance of FFT3D significantly degrades with coarser granularities. As mentioned in the previous section (and shown in Figure 2), in this configuration, only the FFT3D requires a bandwidth close to the maximum installed and, as expected, is the only one that is penalized if the installed bandwidth is not efficiently used. Figure 4(b) shows the bandwidth obtained by the MIC for all applications. It can be seen that for most of the other applications, only when the bandwidth efficiency of the MIC falls below 14GB/s (close to half the installed) the applications present performance variation.

However, as shown in Figure 5(a), when the installed bandwidth is close to the required by most applications —in the case of the 32-processor architecture, 51.2GB/s memory bandwidth is less than the required by Cholesky but more than the required by MatMul— most applications present an important performance degradation. The figure shows speedup against the 8-processor configuration and BHM-4K interleaving, and it is clear that, as the interleaving granularity increases, the applications performance degrades. The MICs' bandwidth, Figure 5(b), shows a similar behavior as the presented for 8 processors, where the granularity increase generates a general bandwidth decrease. Only Knn, that does not require much bandwidth, gets its performance unchanged.

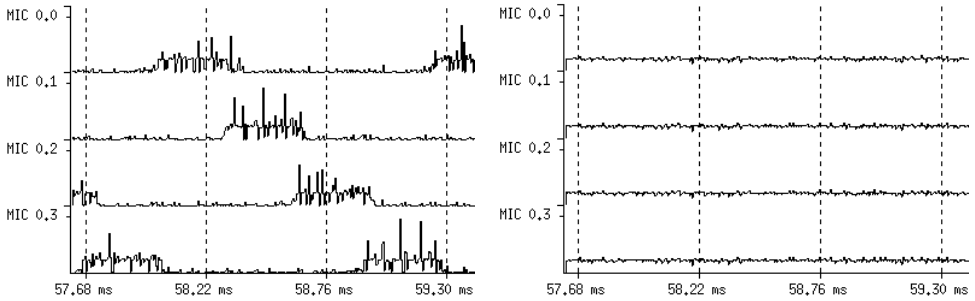
While in the 8-processor architecture, the BHM-128' MIC bandwidth is the highest for all applications, in the case of 32 processors, some applications have very similar bandwidth from BHM-128 until BHM-2K. This is due to the increase of traffic that reaches the DRAM channels, given that in the first case there are 2 processors per DRAM while in the second case there are 4 processors per channel.

In the case of FFT3D with 32 processors, when BHM-16K is used, the performance suffers a $2\times$ slowdown than the one with BHM-128. But, what is more surprising is that even though it has 4 times more processors and twice the installed bandwidth than the experiment with 8 processors, it is very close to the one with 8 processors and BHM-128.

Finally, it can be observed from Figure 4(b) that when the installed bandwidth is enough for the applications its bandwidth usage, from fine- to coarse-grain interleaving, can

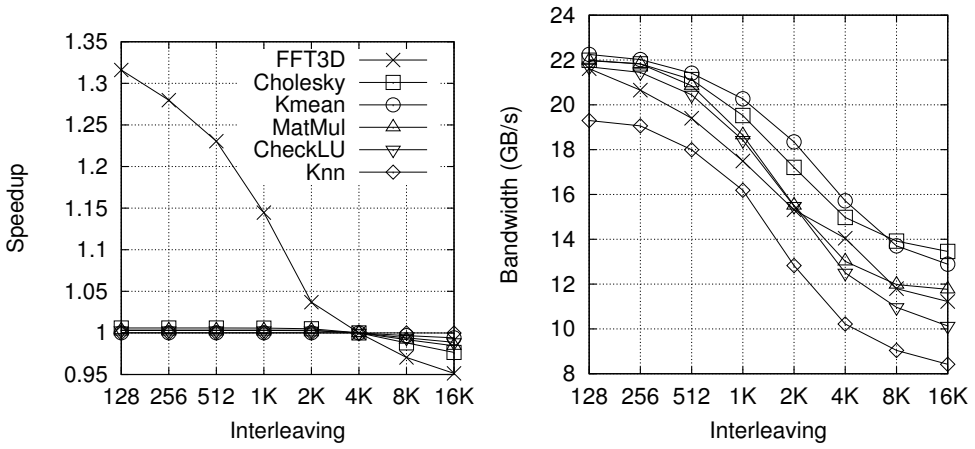
Interleaving	FFT1D	Transposition	FFT1D	Transposition	FFT1D	total
BHM-128	20.1ms	14.7ms	20.1ms	14.1ms	20.2ms	89.2ms
BHM-4096	20.5ms	15.4ms	20.7ms	40.0ms	20.8ms	117.4ms

TABLE IV
FFT3D TIMINGS BREAK DOWN.



(a) Interleaving BHM-4096. (b) Interleaving BHM-128.

Fig. 3. Second transposition request trace of the FFT3D for the MIC DRAM channels, for 25.6GB/s peak bandwidth (4 DRAM channels of 6.4 GB/s). The graph for MIC 0.D means the number of requests for channel D in MIC 0 over time.



(a) Speedup. (b) MIC's effective bandwidth.

Fig. 4. Simulation results BHM-x interleavings for 8 processors with 1 MIC with 4 DRAM channels of 6.4 GB/s each. Speedup is measured against the 8 processors and BHM-4096 interleaving.

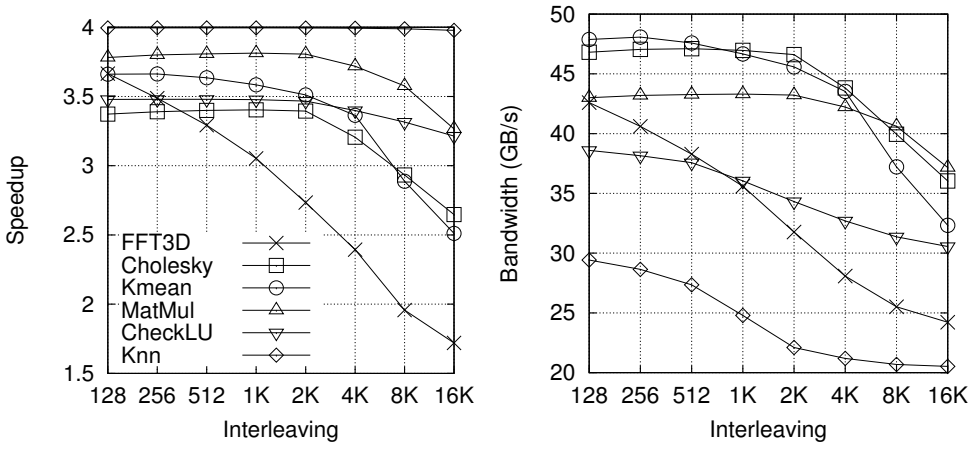
decrease from 50% to 40% of the installed peak bandwidth. However, in Figure 5(b) when the installed bandwidth is close to the bandwidth required by applications, the bandwidth-efficiency use can decrease from 85% to 50% of the installed peak bandwidth, which can translate in 2× slowdown (Figure 5(b)).

VI. CONCLUSIONS

The current trend towards CMP architectures is increasing the memory system bandwidth requirements further than what a single off-chip channel can offer, in order to feed data to all the on-chip processors. The solution is to increase the number of DRAM channels per chip. The main concern of memory controllers for uni-processor systems with few DRAM channels, used to be bank conflicts, since this determines MIC performance. However, with current multi-cores or many-cores requiring many DRAM channels the number of banks increases (for example, a 4-channel system with 8 banks each, contains 32 banks), which reduces the possibility of bank conflicts, and reduces its importance. We show that the main concern of multi-channel DRAM memory systems should be the interleaving granularity given that this determines the number of requests that can reach the channels in parallel and, therefore, the achievable bandwidth of the system.

We have explored a range of memory storage schemes, and have concluded that the most relevant factor in such storage schemes is how frequently data accesses change from one DRAM channel to the next. The interleaving frequency of 128 bytes is higher than the 4KB standard operating system page, meaning that the page allocator in the OS can not decide in which channel to map a page, because all OS pages are spread across many channels. This high interleaving frequency requirement implies that the number of channels must always be a power of 2, since all bit combinations must be valid.

Intuition, and previous works show that fine grain interleaving would perform better than coarse grain ones, in terms of memory bandwidth usage. The objective of this paper



(a) Speedup (b) MICs' effective bandwidth.

Fig. 5. Simulation results BHM-x interleavings for 32 processors with 2 MICs with 4 DRAM channels of 6.4 GB/s each. Speedup is measured against the 8 processors and BHM-4096 interleaving.

was not to propose the use of fine grain interleaving but to show the increasing importance of its use and, mainly, present the degree of performance degradation that can occur. When applications have enough bandwidth, their bandwidth usage (from fine- to coarse-grain interleaving) can decrease from 50% to 40% of the installed peak bandwidth. However, when the bandwidth required is very close to the installed bandwidth, the bandwidth-efficiency use can decrease from 85% to 50% of the installed peak bandwidth, which can translate in $2\times$ slowdown. We showed that the degradation is due to the dynamic unbalance caused by the access pattern to the memory channels.

ACKNOWLEDGMENT

The authors want to thank Carlos Villavieja, Milan Pavlovic, Toni Quesada and Pieter Bellens for their contributions to the development of the simulator employed for this work.

REFERENCES

[1] *More chip cores can mean slower supercomputing*, Sandia National Laboratories, http://www.sandia.gov/news/resources/releases/2009/multicore.html, January 2009.

[2] H. P. Hofstee, "Power efficient processor architecture and the Cell processor," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005, pp. 258–262.

[3] R. Badia, J. Perez, E. Ayguade, and J. Labarta, "Impact of the memory hierarchy on shared memory architectures in multicore programming models," in *Proceedings of Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Feb. 2009, pp. 437–445.

[4] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: an efficient multithreaded runtime system," *SIGPLAN Not.*, vol. 30, no. 8, pp. 207–216, 1995.

[5] M. D. McCool, K. Wadleigh, B. Henderson, and H.-Y. Lin, "Performance evaluation of gpus using the rapidmind development platform," in *Proceedings of the ACM/IEEE conference on Supercomputing*, Nov. 2006.

[6] K. Fatahalian, T. J. Knight, M. Houston, M. Erez, D. R. Horn, L. Leem, J. Y. Park, M. Ren, A. Aiken, W. J. Dally, and P. Hanrahan, "Sequoia: Programming the memory hierarchy," *Proceedings of the ACM/IEEE Supercomputing Conference*, Nov. 2006.

[7] K. Stavrou, M. Nikolaides, D. Pavlou, S. Arandi, P. Evripidou, and P. Trancoso, "Tflux: A portable platform for data-driven multithreading on commodity multicore systems," in *ICPP '08: Proceedings of the 2008 37th International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 25–34.

[8] P. Budnik and D. Kuck, "The organization and use of parallel memories," *Computers, IEEE Transactions on*, vol. C-20, no. 12, pp. 1566–1569, Dec. 1971.

[9] W. Frailong, J. M. Jalby, and J. Lenfant, "Xor-schemes: A flexible data organization in parallel memories," in *Proceedings of the International Conference on Parallel Processing*, August 1985, pp. 276–283.

[10] D. T. Harper III and D. A. Linebarger, "A dynamic storage scheme for conflict-free vector access," *SIGARCH Comput. Archit. News*, vol. 17, no. 3, p. 7277, 1989.

[11] M. Valero, T. Lang, J. M. Llaberia, M. Peiron, Navarro, J. J., and E. Ayguade, "Conflict-free strides for vectors in matched memories," *Parallel Processing Letters*, vol. 1, no. 2, pp. 95–102, 1991. [Online]. Available: citeseer.ist.psu.edu/article/valero92conflictfree.html

[12] M. Valero, T. Lang, J. M. Llabería, M. Peiron, E. Ayguadé, and J. J. Navarra, "Increasing the number of strides for conflict-free vector access," in *ISCA '92: Proceedings of the 19th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 1992, p. 372381.

[13] M. Peiron, M. Valero, and E. Ayguadé, "Synchronized access to streams in simd vector multiprocessors," in *ICS '94: Proceedings of the 8th international conference on Supercomputing*. New York, NY, USA: ACM, 1994, p. 2332.

[14] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, 2000, pp. 32–41.

[15] W.-F. Lin, S. Reinhardt, and D. Burger, "Reducing dram latencies with an integrated memory hierarchy design," in *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, 2001, pp. 301–312.

[16] J. H. Zurawski, J. E. Murray, and P. J. Lemmon, "The design and verification of the alphastation 600 5-series workstation," *Digital Tech. J.*, vol. 7, no. 1, p. 8999, 1995.

[17] V. Cuppu and B. Jacob, "Organizational design trade-offs at the dram, memory bus, and memory controller level: Initial results," University of Maryland Systems and Computer Architecture Group, Tech. Rep. UMD-SCA-TR-1999-2, November 1999.

[18] B. Jacob, S. W. Ng, and D. T. Wang, *Memory systems: cache, DRAM, disk*. 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA: Morgan Kaufmann Publishers, 2008.

[19] *Intel 955X Express Chipset*, Intel, April 2005.

[20] M. Peiron, M. Valero, E. Ayguade, and T. Lang, "Vector multiprocessors with arbitrated memory access," in *Computer Architecture, 1995.*

Proceedings. 22nd Annual International Symposium on, 1995, pp. 243–252. [Online]. Available: citeseer.ist.psu.edu/peiron95vector.html

[21] S. A. McKee, "Hardware support for dynamic access ordering: Performance of some design options," University of Virginia, Charlottesville, VA, USA, Tech. Rep., 1993.

[22] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory access scheduling," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, 2000, pp. 128–138.

[23] S. Rixner, "Memory controller optimizations for web servers," in *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on*, Dec. 2004, pp. 355–366.

[24] E. Ipek, O. Mutlu, J. Martinez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, June 2008, pp. 39–50.

[25] J. Corbal, R. Espasa, and M. Valero, "Command vector memory systems: high performance at low cost," in *Parallel Architectures and Compilation Techniques, 1998. Proceedings. International Conference on*, Oct 1998, pp. 68–77.

[26] S. McKee and W. Wulf, "A memory controller for improved performance of streamed computations on symmetric multiprocessors," in *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International*, Apr 1996, pp. 159–165.

[27] S. McKee, W. Wulf, J. Aylor, R. Klenke, M. Salinas, S. Hong, and D. Weikle, "Dynamic access ordering for streamed computations," *Computers, IEEE Transactions on*, vol. 49, no. 11, Nov 2000.

[28] V. Cuppu and B. Jacob, "Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor dram-system performance?" in *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, 2001, pp. 62–71.

[29] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, "Ibm power6 microarchitecture," *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 639–662, November 2007.

[30] A. Rico, F. Cabarcas, A. Quesada, M. Pavlovic, A. J. Vega, C. Villavieja, Y. Etsion, and A. Ramrez, "Scalable simulation of decoupled accelerator architectures," Universitat Politecnica de Catalunya, Tech. Rep. UPC-DAC-RR-2010-14, June 2010.